Distributed Ray Tracing

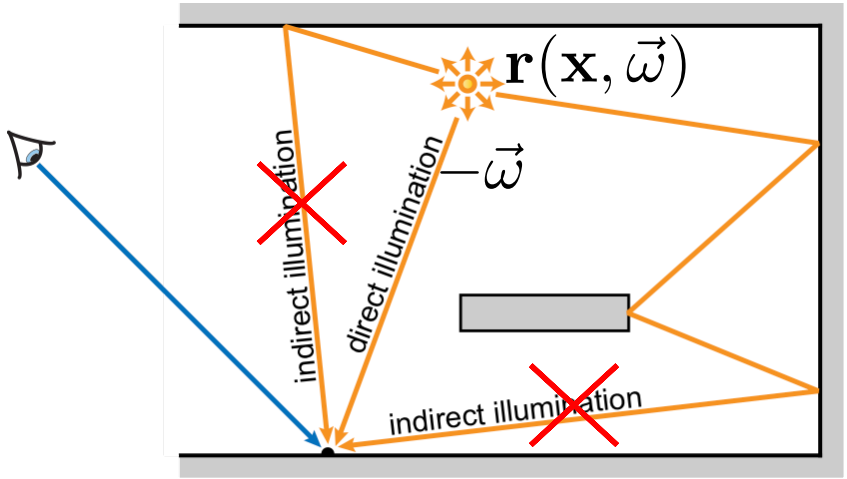
Dr Cengiz Öztireli

Direct Illumination

How do we use the rendering equation in practice? One way is local or direct illumination.
In this model, light can only come from light sources, i.e. we ignore light reflected from a surface and landing on another surface.
This is useful for fast renderings and is what rasterizers such as the ones in OpenGL typically assume.

- All light comes directly from emitters, i.e. light sources

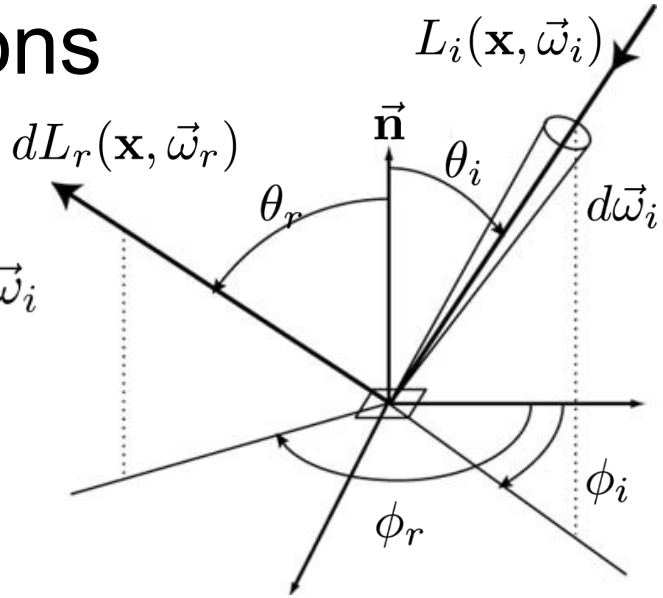
$$L_o(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$



$$L_i(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{r}(\mathbf{x}, \vec{\omega}), -\vec{\omega})$$

- The reflected radiance due to incident illumination from all directions

$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$



Problem: estimating this integral

Quadrature: estimating integrals

Such estimations can be performed by sampling the function to be integrated and summing up the values.
This is called quadrature. There is a resulting estimator for the actual value of the integral.
The samples are typically taken from an underlying probability distribution function. In order to make sure we get the right integral value on average (i.e. we get an unbiased estimator), we need to normalize the sampled values.

- Importance sampling

function to integrate

$$\langle F^N \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

estimator

#samples

density of samples

Applying this idea to the integral we have, we get the following form.
The samples are direction samples, i.e. we sample the hemisphere over which we are integrating.

- Importance sampling

$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

$$\langle L_r(\mathbf{x}, \vec{\omega}_r)^N \rangle = \frac{1}{N} \sum_{k=1}^{N} \frac{f_r(\mathbf{x}, \vec{\omega}_{i,k}, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_{i,k}) \cos\theta_{i,k} \, d\vec{\omega}_{i,k}}{p_\Omega(\vec{\omega}_{i,k})}$$

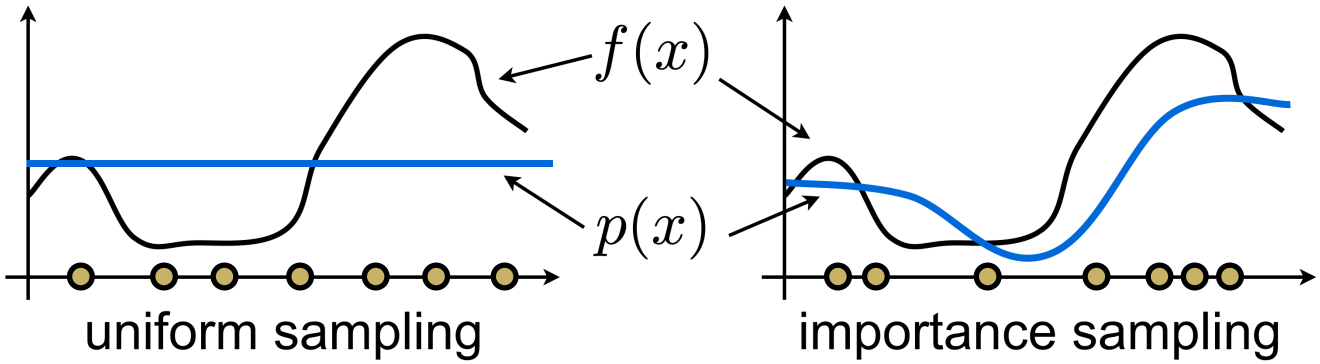
estimator #samples density of samples

Quadrature: estimating integrals

The idea behind importance sampling is simple: place more samples where the function value is high. This is in contrast with uniform sampling, where all samples are uniformly distributed over the domain.

- Importance sampling

$$\langle L_r(\mathbf{x}, \vec{\omega}_r)^N \rangle = \frac{1}{N} \sum_{k=1}^{N} \frac{f_r(\mathbf{x}, \vec{\omega}_{i,k}, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_{i,k}) \cos\theta_{i,k} \, d\vec{\omega}_{i,k}}{p_\Omega(\vec{\omega}_{i,k})}$$



Sampling Terms

Our function to be integrated consists of multiple terms. Ideally, we want to sample their product.
But this is typically not possible. Note that knowing a function does not mean we can efficiently sample from it.
The idea is then to choose what to sample. A candidate is the cosine term.

- What to importance sample?

$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

Sampling the Cosine Term

A case where sampling the cosine term is a good idea is when the BRDF f and lighting function L_i are constants.
This is true when we have diffuse objects (constant BRDF) illuminated by the sky (constant L_i).
In this case, we are left with an integral of multiplication of the cosine term and the visibility function.
We will see what visibility function is in the next pages.

- Example: diffuse objects illuminated by an ambient white sky (ambient occlusion)

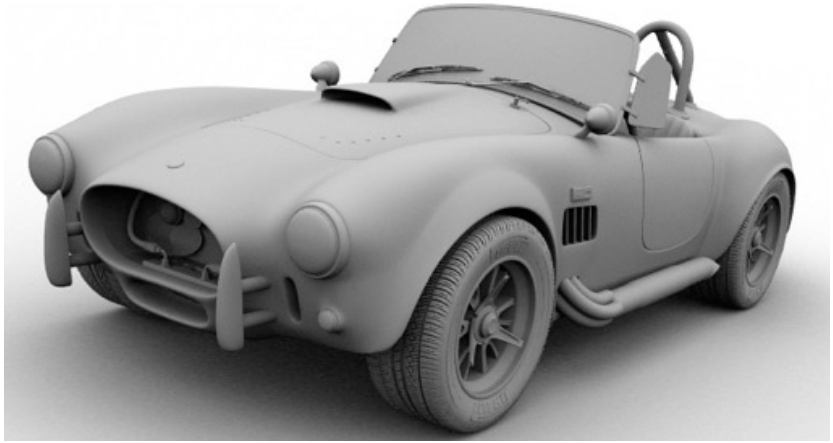
$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

$$L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

visibility function

This is also called ambient occlusion and an important way to render objects.

- Ambient occlusion examples



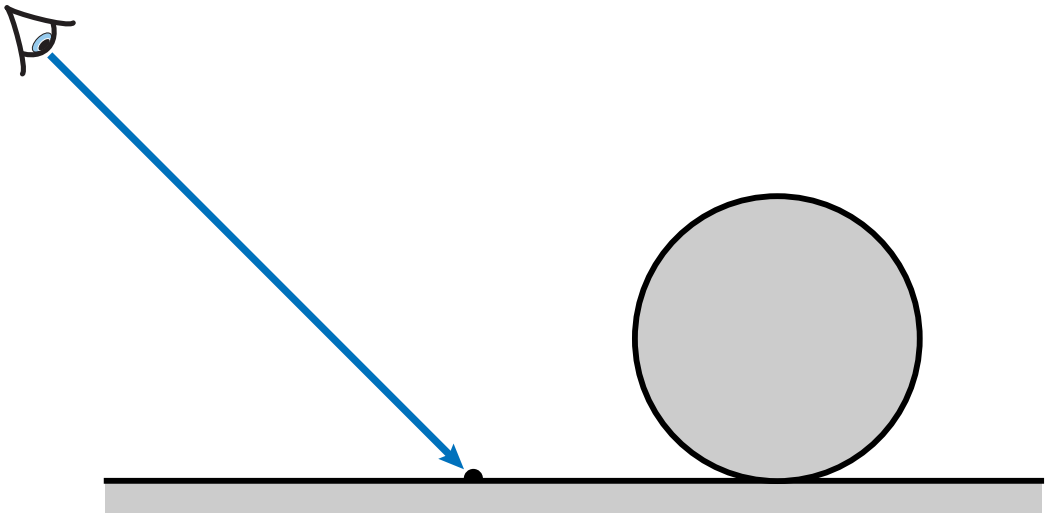
Sampling the Cosine Term

So how do we sample to approximate this integral?

Imagine we are looking at a point in the scene for which we want to compute the approximation.

Recall that the reflected light is constant with respect to the outgoing light direction in this case.

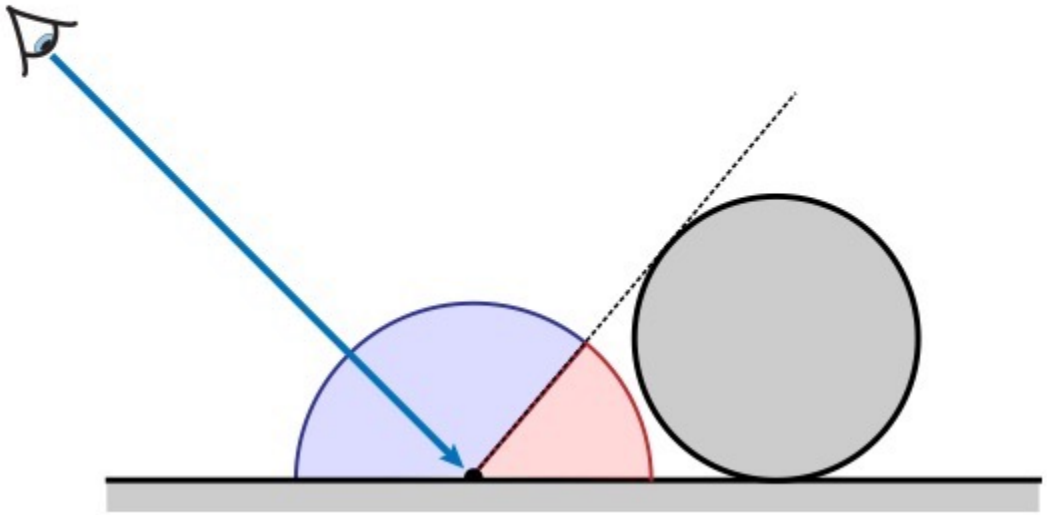
- Ambient occlusion $L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$



Sampling the Cosine Term

Only some of the light from the scene will reach the point as some is blocked by the object (red region). This is captured by the visibility term. The visibility function V is a binary function (either 1 or 0) indicating light can reach the point or not.

- Ambient occlusion $L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$



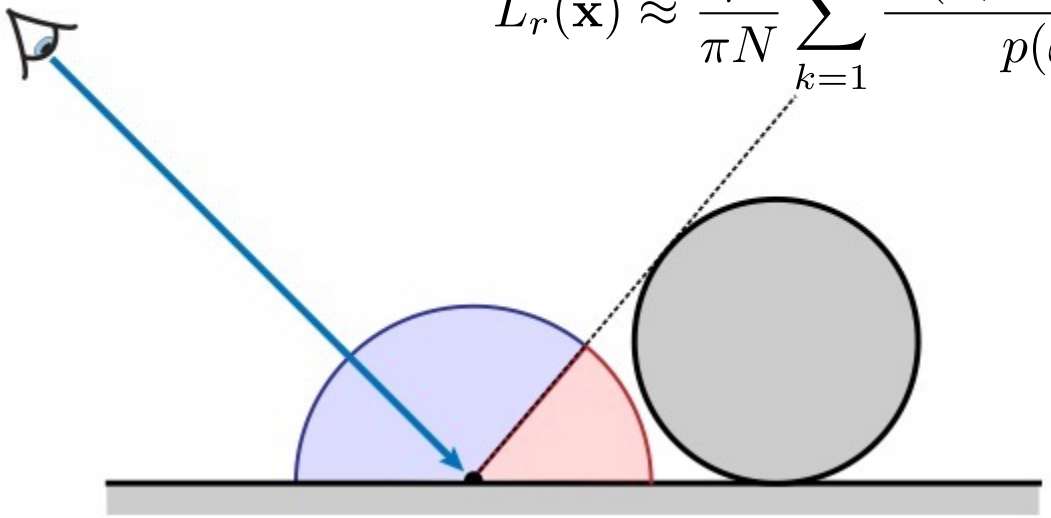
Sampling the Cosine Term

For each sample direction, we get a term in the sum. The task is to define what the distribution p is. This is the distribution of the direction samples indexed by k in the sum.

- Ambient occlusion

$$L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

$$L_r(\mathbf{x}) \approx \frac{\rho}{\pi N} \sum_{k=1}^{N} \frac{V(\mathbf{x}, \vec{\omega}_{i,k}) \cos\theta_{i,k}}{p(\vec{\omega}_{i,k})}$$

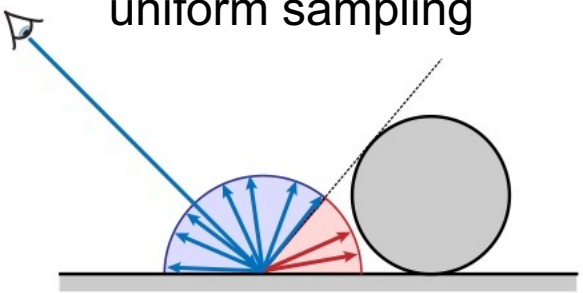


Sampling the Cosine Term

Uniform sampling means there is no extra weighting by p and we randomly distribute the directions.
This is not good as we are wasting samples for which the cosine term is close to zero.
A better approach is importance sampling with the cosine term. This ensures the samples that contribute more are drawn more.

- Ambient occlusion $L_r(\mathbf{x}) \approx \frac{\rho}{\pi N} \sum_{k=1}^{N} \frac{V(\mathbf{x}, \vec{\omega}_{i,k}) \cos\theta_{i,k}}{p(\vec{\omega}_{i,k})}$

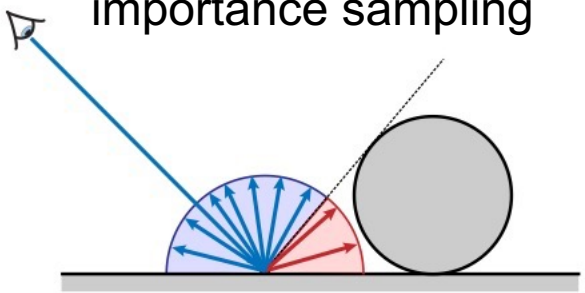
uniform sampling



$p(\vec{\omega}_{i,k}) = 1/2\pi$

$L_r(\mathbf{x}) \approx \frac{2\rho}{N} \sum_{k=1}^{N} V(\mathbf{x}, \vec{\omega}_{i,k}) \cos\theta_{i,k}$

importance sampling



$p(\vec{\omega}_{i,k}) = \cos\theta_{i,k}/\pi$

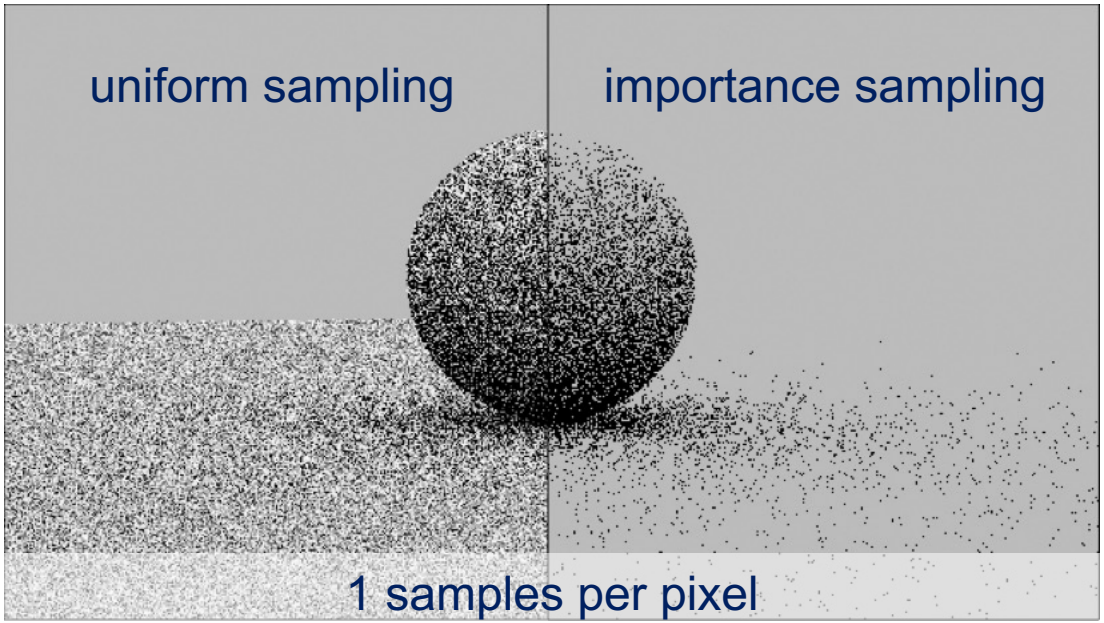
$L_r(\mathbf{x}) \approx \frac{\rho}{N} \sum_{k=1}^{N} V(\mathbf{x}, \vec{\omega}_{i,k})$

Sampling the Cosine Term

It is very important to perform importance sampling for less noisy renderings.

Importance sampling is standard in all rendering frameworks. The idea extends well beyond rendering.

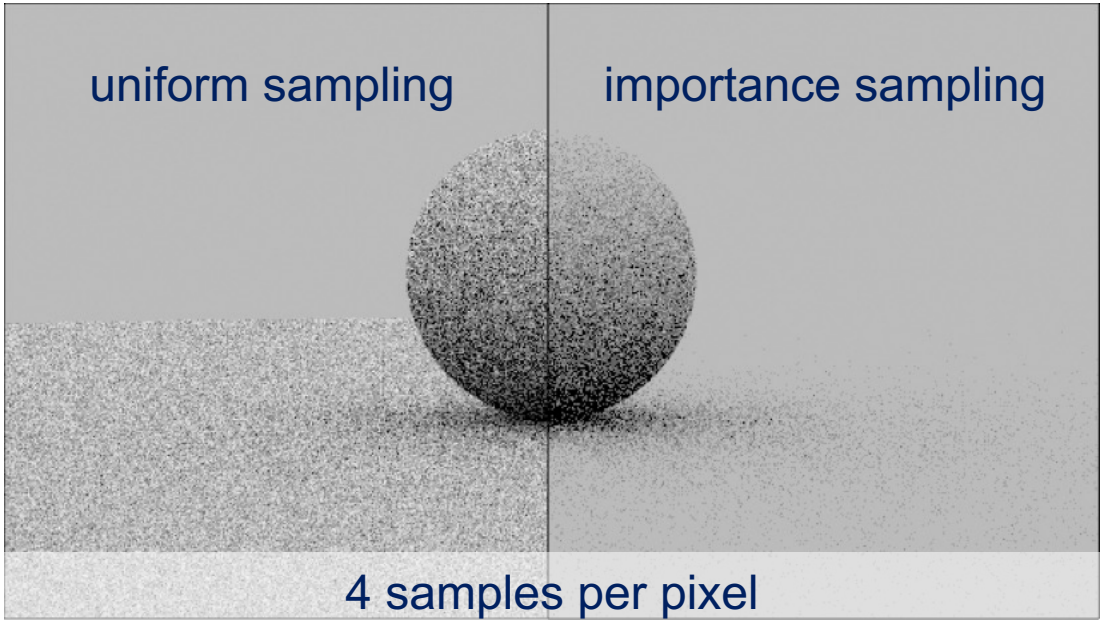
- Ambient occlusion



Sampling the Cosine Term

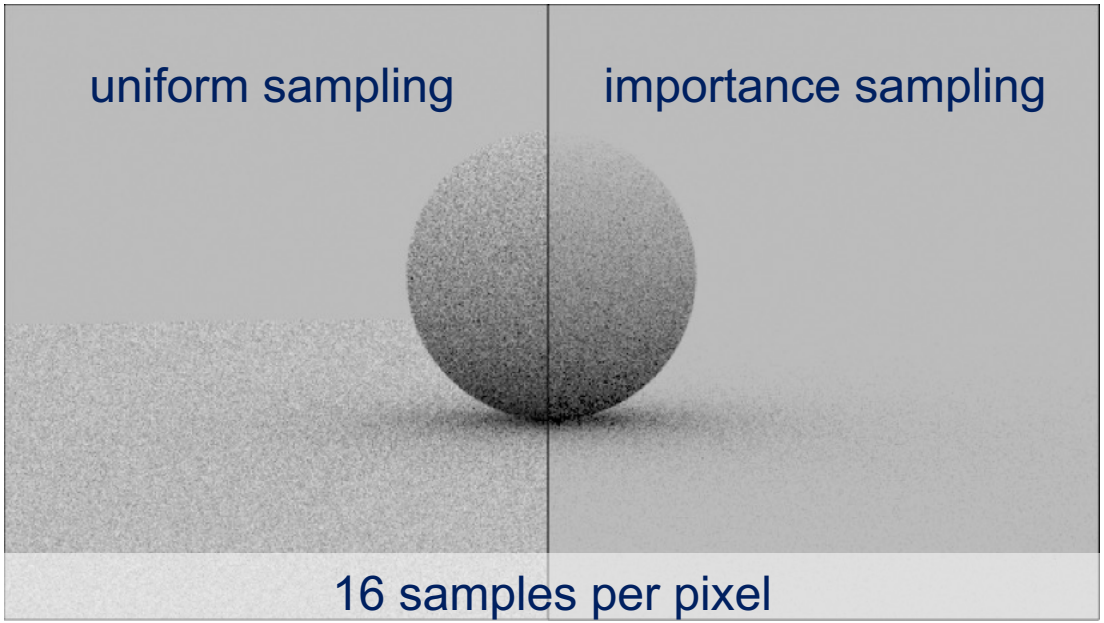
We start to see the significance of importance sampling as the renderings become more reasonable with more samples.

- Ambient occlusion



Sampling the Cosine Term

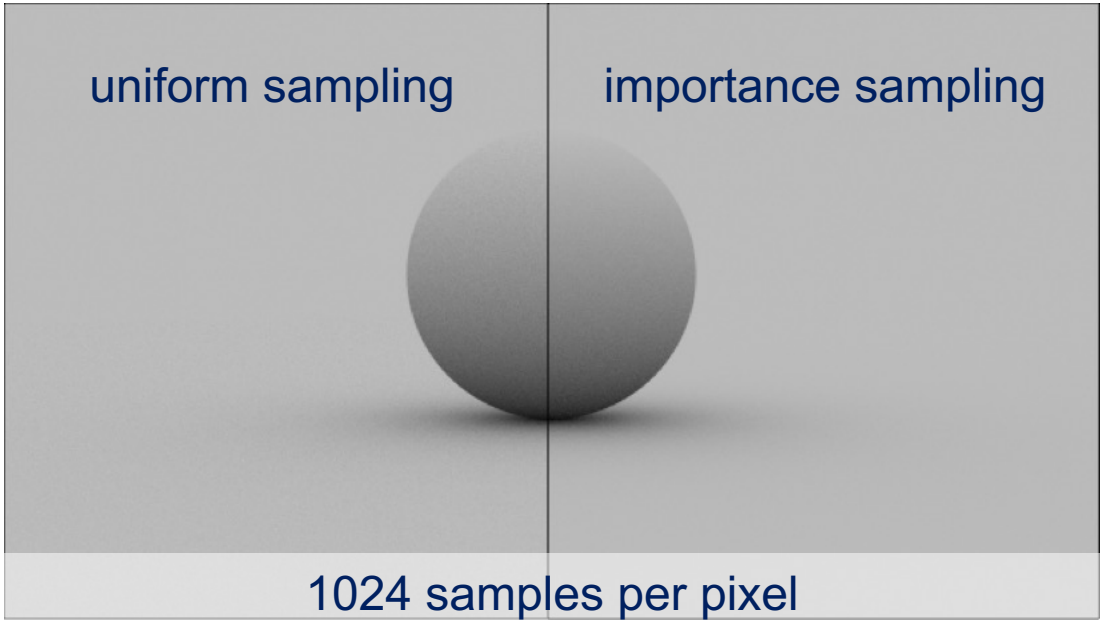
- Ambient occlusion



Sampling the Cosine Term

For high enough sample counts, we can get a similar rendering with any unbiased sampling scheme, e.g. uniform sampling.

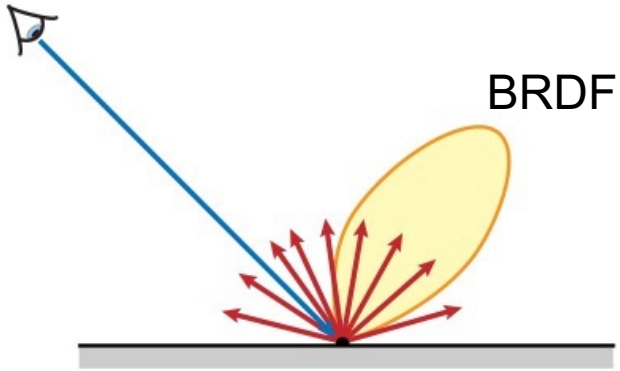
- Ambient occlusion



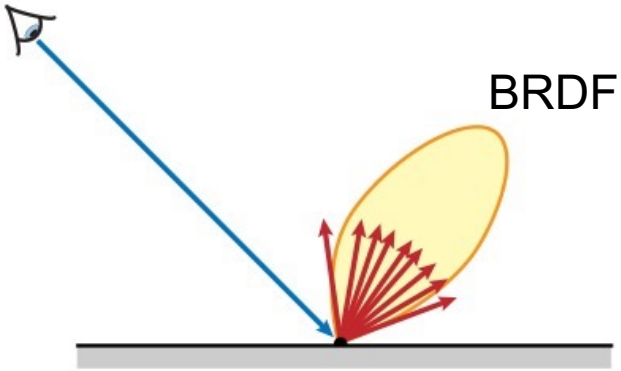
Sampling BRDFs

In general, BRDF functions will not be constant and sometimes it is better to importance sample with them. This is typically true for highly peaky functions where BRDF is mostly zero or low except at a few samples. With uniform or cosine based sampling, it is highly probable that we will miss the high values of BRDFs.

- What to importance sample?



Cosine-weighted
Importance sampling



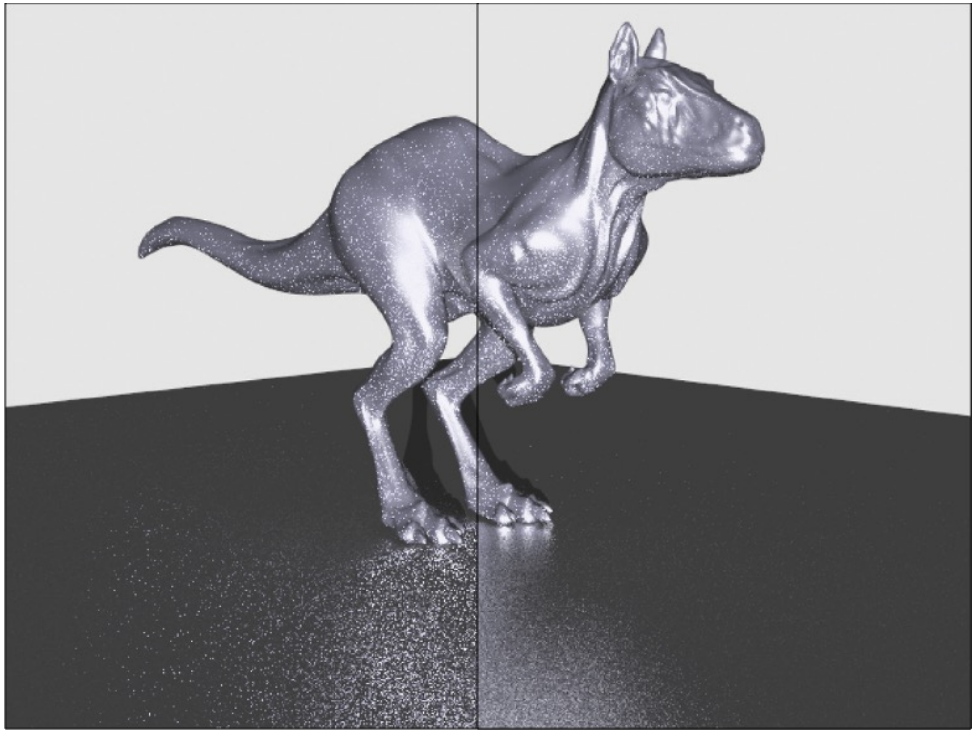
BRDF Importance sampling

$$p(\vec{\omega}_i) \propto f(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r)$$

Sampling BRDFs

You can clearly see the effects of this in renderings.

uniform hemispherical sampling



BRDF importance sampling

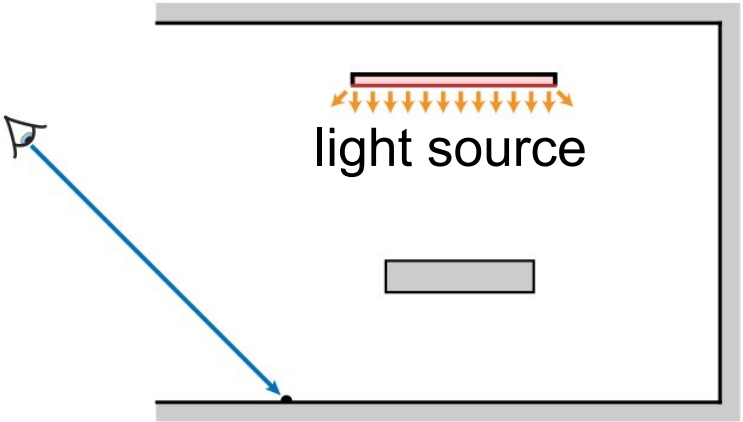
Sampling Lights

A further idea is importance sampling with the lighting function.

This can also be performed by sampling the light sources in the scene. The advantage of this is that we don't have to sample again for each different point in the scene.

- What to importance sample?

$$L_r(\mathbf{x}, \vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$



sample emissive surfaces only

Global Illumination

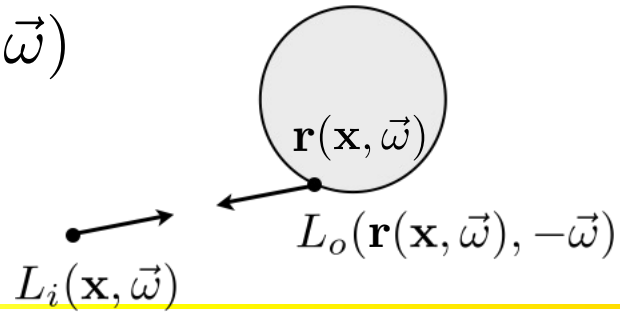
So far we considered a local illumination model: light only comes from light sources.
In general, light reflects off the surfaces and reaches another point in the scene.
We can thus write the incoming light at a point $\mathbf{x}$ equivalently as the outgoing light at some other point $\mathbf{r}$.

- The rendering equation

$$L_o(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

- No participating media → radiance is constant along rays
- We can relate incoming radiance to outgoing radiance

$$L_i(\mathbf{x}, \vec{\omega}) = L_o(\mathbf{r}(\mathbf{x}, \vec{\omega}), -\vec{\omega})$$



This form of the rendering equation makes the recursive nature of rendering clear. Note that L appears on both sides of the expression.

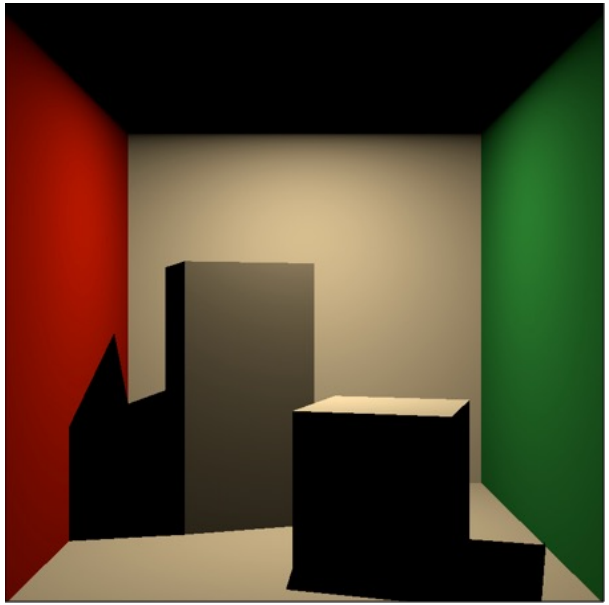
- The rendering equation

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' d\vec{\omega}'$$

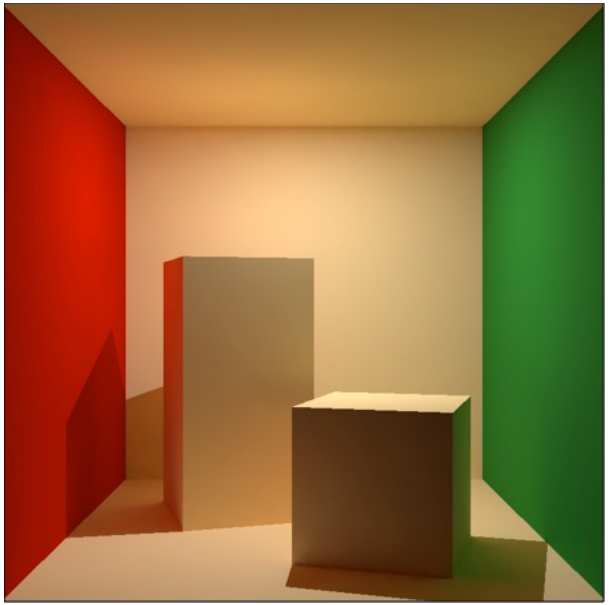
same function as on the left

- Only outgoing radiance functions, we drop o subscript
- Fredholm equation of the second kind (recursive)

Global Illumination



local illumination



global illumination

Global Illumination

- Subsurface scattering

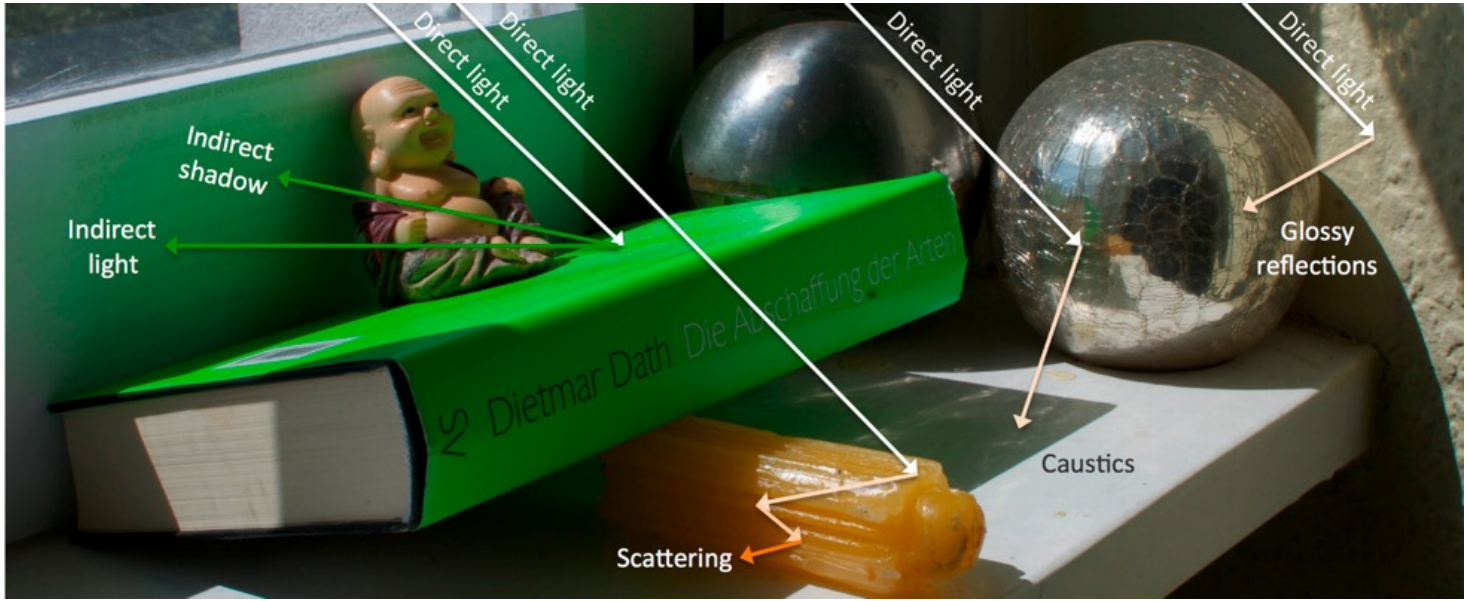
Global Illumination

- Caustics



Global Illumination

- And more



Solving the rendering equation

Ray/ path/ light tracing is an unbiased method to solve this equation as compared to others that we will not talk about.

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' d\vec{\omega}'$$

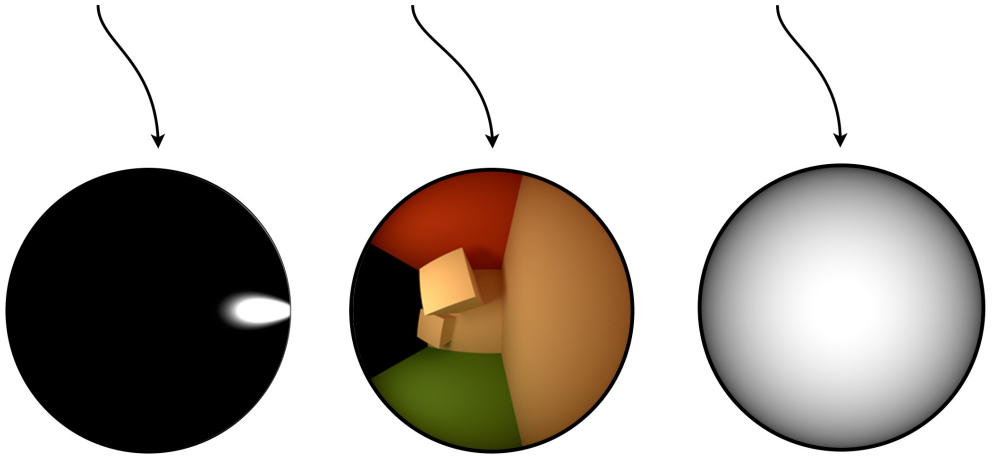
Monte Carlo methods

- **Unbiased methods**
  - Recursive ray tracing
  - Path tracing and light tracing
  - Bidirectional path tracing
- **Biased methods**
  - Many-light algorithms
  - Density estimation
  - Photon mapping
  - Irradiance caching

Solving the rendering equation

In a typical rendering framework, BRDF (f_r) and the cosine term are provided.
They are plotted as functions over the hemisphere here.
The problem is the lighting function L, which depends on light coming from other parts of the scene.

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' d\vec{\omega}'$$



How do we get this?

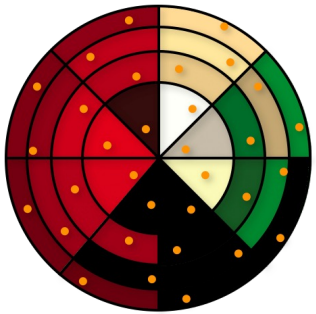
Recursive Ray Tracing

The solution to this equation is also recursive. At each recursion step, a number of direction samples are taken.

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{H^2} f_r(\mathbf{x}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta' d\vec{\omega}'$$

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$

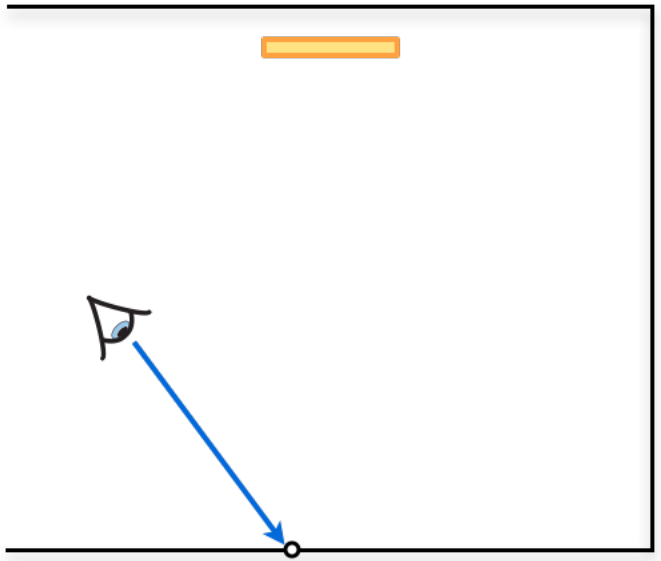
Sample the hemisphere -- recursively



Recursive Ray Tracing

Let's see how this works when we start from the sensor. Each point on the sensor can be imagined as an "eye".
We first send out a ray from the eye and intersect that with the geometry in the scene.
This gives us the point and direction in which we want to evaluate how much light is leaving to reach the eye.

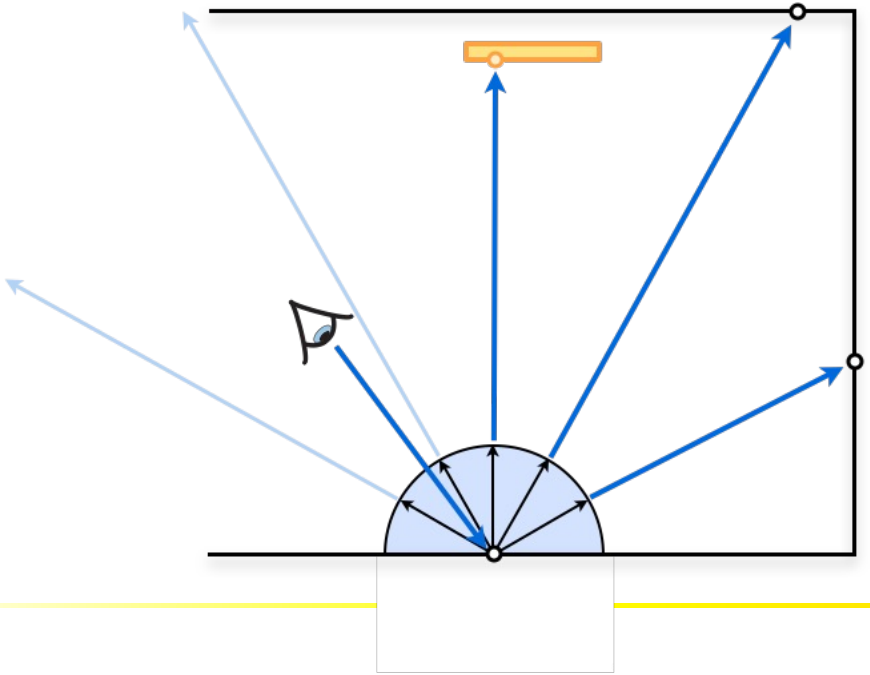
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$



Recursive Ray Tracing

The expression below tells us to take a set of direction samples.
Some of these samples will end up on a light source, for which we already now the lighting function.
But others will land on other surfaces. For those, we don't know the lighting function.

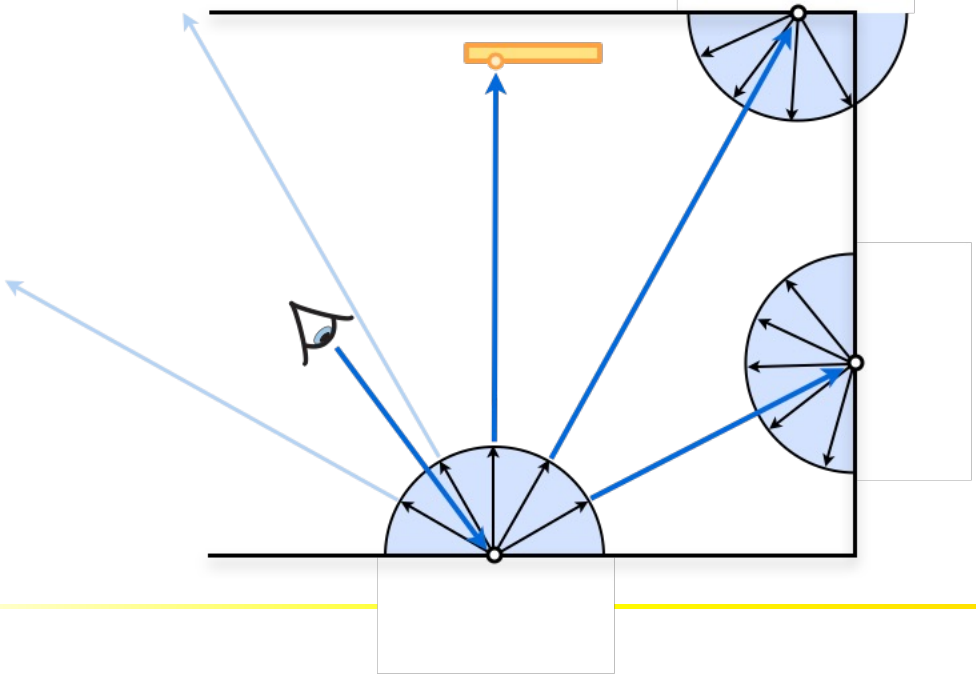
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$



Recursive Ray Tracing

At those points, to evaluate how much light is leaving the surfaces, we need to shoot new rays.

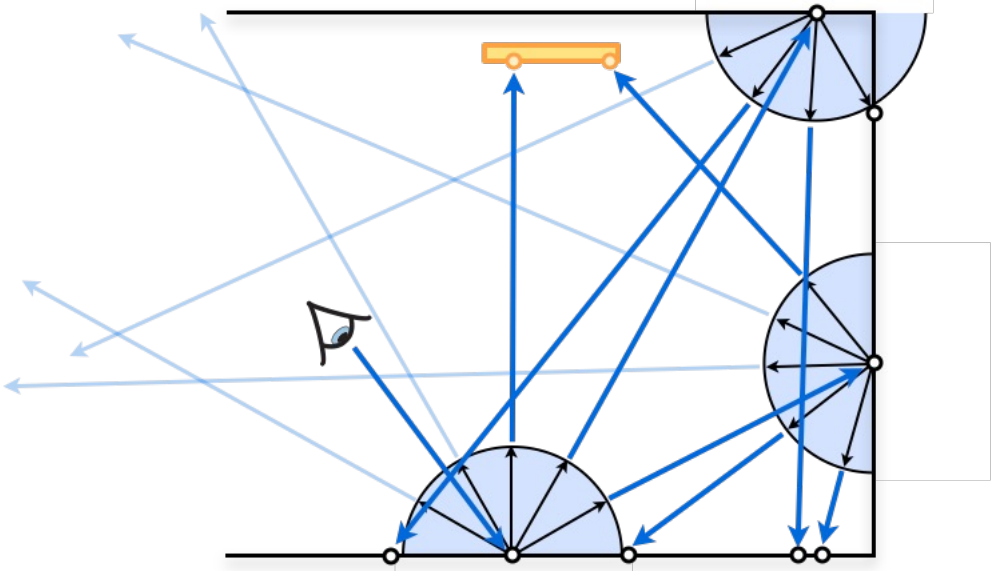
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$



Recursive Ray Tracing

Similar to the previous case, some of these will end up on light sources and some on surfaces.
We thus need to continue recursively until we reach a certain recursion level. In this example, it is two.
Some of the rays are connected to the light source. Those contribute to light at the intermediate points, which in turn contribute to the light at the original point we are interested in.

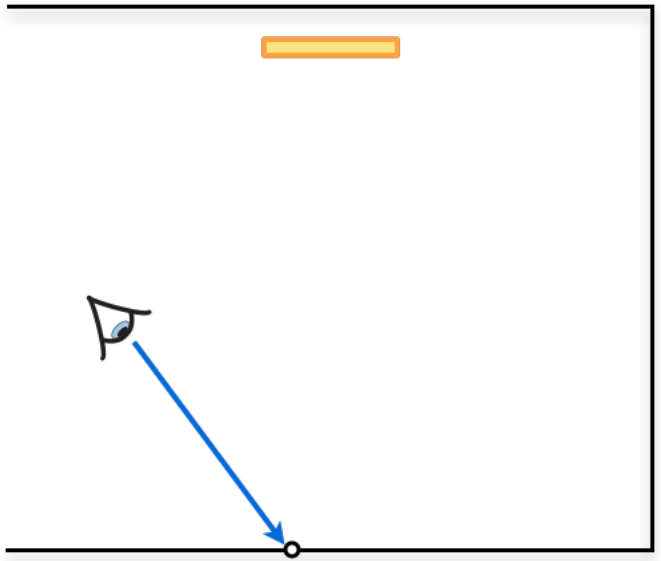
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$



Recursive Ray Tracing

This algorithm can be made more efficient using the co-called shadow rays.

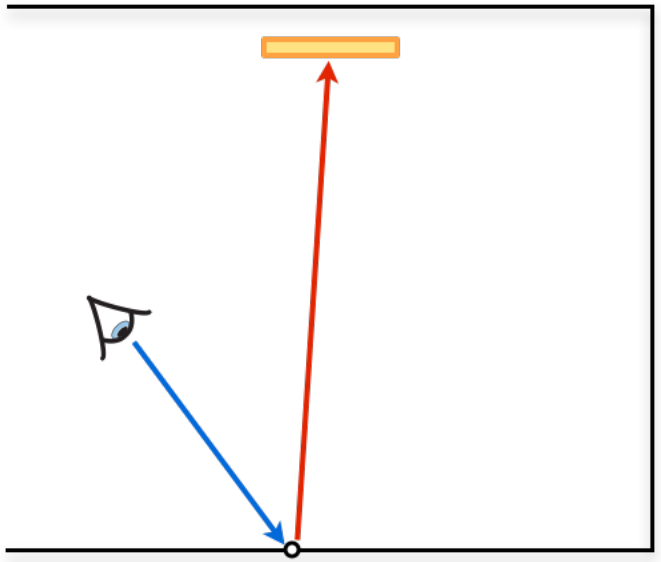
$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Recursive Ray Tracing

A shadow ray is a ray that connects a point in the scene to a light source.

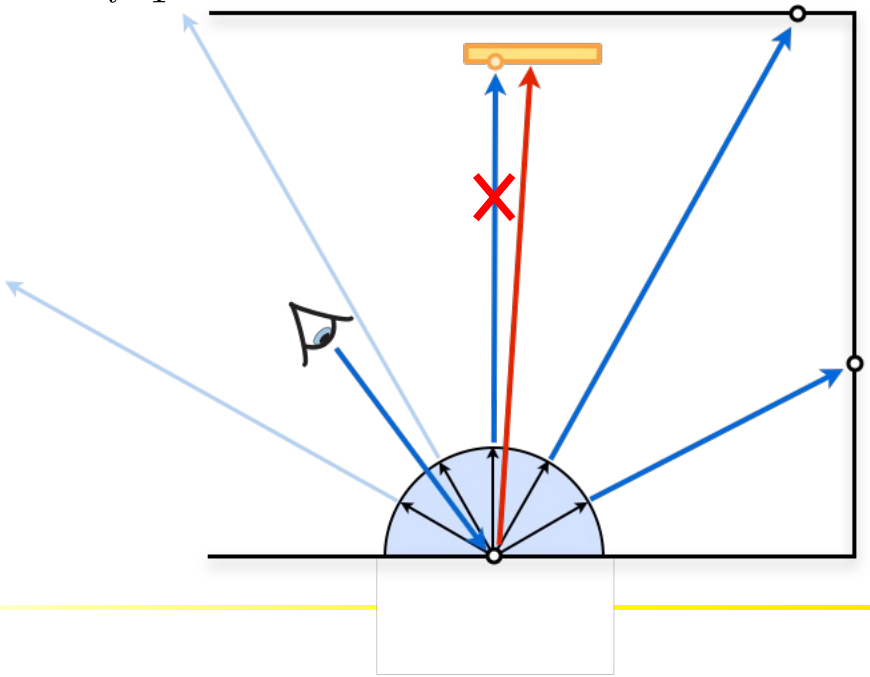
$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Recursive Ray Tracing

A shadow ray is a ray that connects a point in the scene to a light source.
The contribution of light coming from light sources is thus taken care of by shadow rays.
We terminate the rays that end up on light sources during the recursion to avoid counting the contributions from light sources twice.

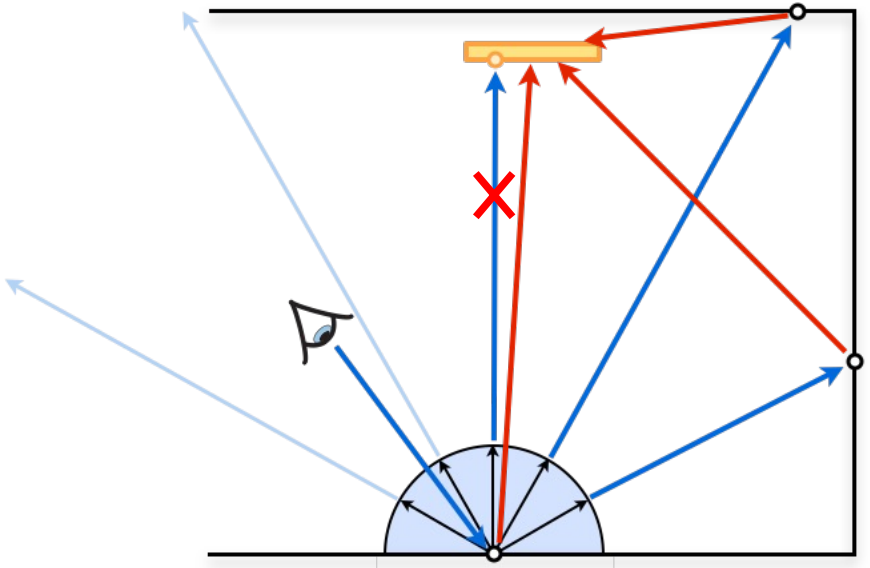
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Recursive Ray Tracing

Shadow rays are sent at each recursion step.

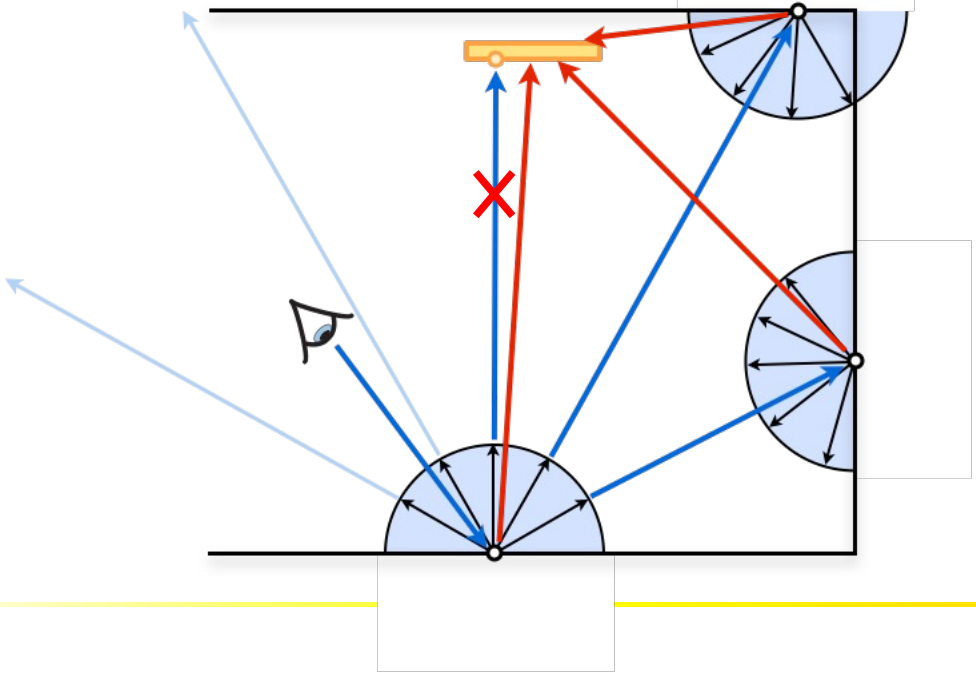
$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Recursive Ray Tracing

Recursion proceeds as normal apart from sending these additional rays.

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$ with shadow rays

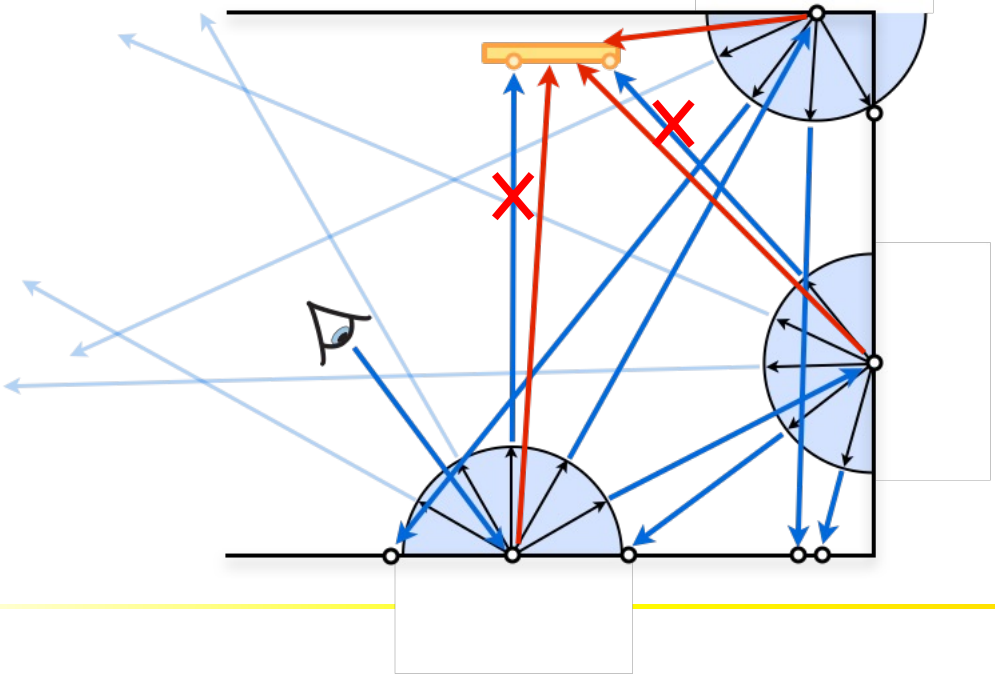


Recursive Ray Tracing

Recursion proceeds as normal apart from sending these additional rays.

Again, we cancel the contributions coming from non-shadow rays that intersect light sources.

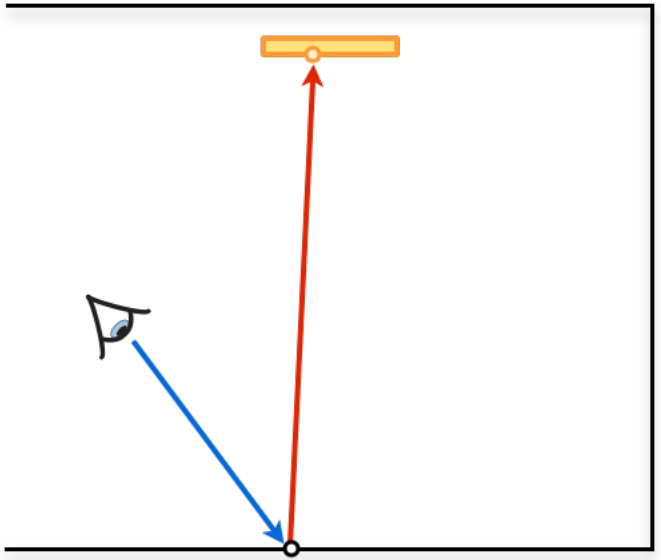
$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N}\frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Path Tracing

Recursive ray tracing is typically not used in practice.
Instead, path tracing, which is conceptually equivalent but computationally more efficient, is utilised.
In path tracing, we again start from the eye, but continue on a single path until we reach a light source.
In this first case, the path has only two segments.

$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays

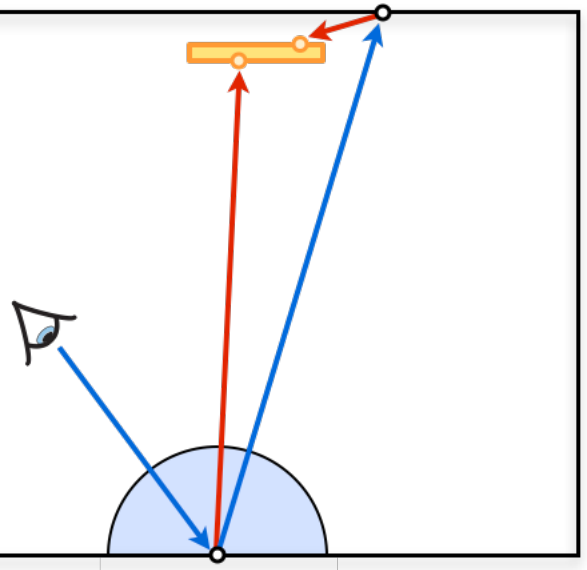


Path Tracing

This second path is a bit more complicated with three segments.

Note that the last segment is always what we called a shadow ray before.

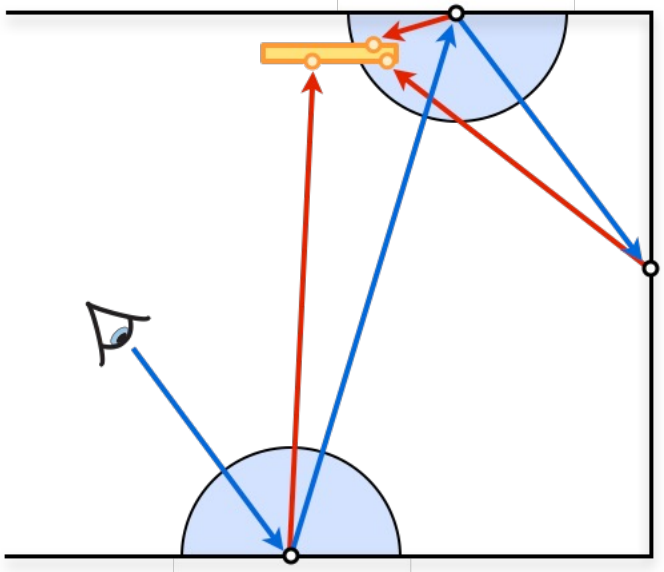
$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Path Tracing

We can define how many bounces of light, i.e. segments, we want and terminate the path accordingly.

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\vec{\mathbf{x}}, \vec{\omega}', \vec{\omega}) L(\mathbf{r}(\mathbf{x}, \vec{\omega}'), -\vec{\omega}') \cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



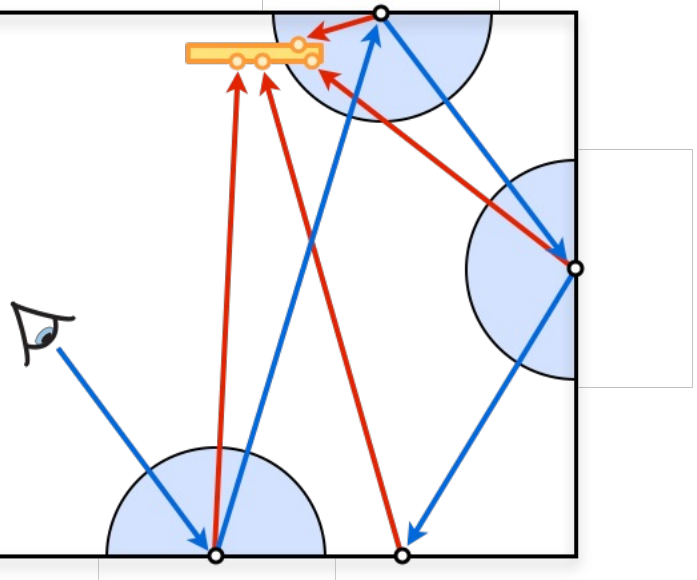
Path Tracing

The contributions coming from all of these n paths are summed up.

Note that in the recursive ray tracing case, we were first estimating the outgoing light at the intermediate points, and then summing up those, recursively. In this case, we only have a single summation.

It can be shown that both path tracing and recursive ray tracing simulate light without bias.

$$L(\mathbf{x},\vec{\omega}) \approx L_e(\mathbf{x},\vec{\omega}) + \frac{1}{N}\sum_{i=1}^{N}\frac{f_r(\vec{\mathbf{x}},\vec{\omega}',\vec{\omega})L(\mathbf{r}(\mathbf{x},\vec{\omega}'),-\vec{\omega}')\cos\theta'}{p(\vec{\omega}')}$$ with shadow rays



Path Tracing

- Full solution to the rendering equation
- Simple to implement
- Slow convergence
    - requires 4x more samples to half the error
- Robustness issues
    - does not handle some light paths well, e.g. caustics
    - no reuse or caching of computation
- General sampling issue
    - makes only locally optimal decisions

Path Tracing

Nature ~ **2 . 10^{33}** / second

Fastest GPU ray tracer ~ **2 . 10^{8}** / second